Uniwersytet w Białymstoku

Wydział Matematyki i Informatyki

Instytut Informatyki

Menedżer piłkarski – gra strategiczna z elementami sztucznej inteligencji

---

Praca licencjacka

Karol Kiersnowski

Promotor:
dr inż. Wojciech Lesiński

Białystok, wrzesień 2015

Spis treści

Wstęp

Od kilku lat interesuję się programowaniem. Wybrałem studia informatyczne głównie z tego powodu. Wybór pracy dyplomowej uzależniałem od tego, aby było to napisanie aplikacji komputerowej. Zastanawiając się nad wyborem tematu, brałem pod uwagę to, żeby napisanie programu wymagało rozległego korzystania z wiedzy, którą nabyłem w trakcie studiowania. Chciałem napisać grę komputerową. Ciekawym pomysłem wydawało mi się napisanie menedżera piłkarskiego. Taka gra korzystałaby między innymi z bazy danych oraz wymagałaby wykorzystania elementów sztucznej inteligencji.

Celem pracy było stworzenie prostej komputerowej gry strategicznej – menedżera piłkarskiego. Od strony technicznej jest to aplikacja okienkowa działająca pod systemem Windows, korzystająca z relacyjnej bazy danych i wykorzystująca elementy sztucznej inteligencji. Oprócz tego, w części teoretycznej praca omawia tematykę gier komputerowych. W pracy zostają przedstawione podstawy tworzenia gier komputerowych.

W rozdziale 1. omówiłem tematykę gier komputerowych. Szczególną uwagę zwróciłem na gry strategiczne. Dokonałem przeglądu podgatunku strategii – menedżerów piłkarskich. W następnym rozdziale przedstawiłem podstawy tworzenia gier komputerowych. Zostały w nim omówione etapy produkcji gier. Następnie przedstawiłem technologie informatyczne wykorzystywane do tworzenia prostych gier komputerowych. Zostały omówione najczęściej stosowane języki programowania, biblioteki, środowiska programistyczne oraz inne narzędzia wykorzystywane przy tworzeniu gier. Rozdział 3. jest opisem implementacji gry napisanej przeze mnie specjalnie na potrzeby pracy. Zostały w nim przedstawione: koncepcja gry, wykorzystane technologie informatyczne, struktura aplikacji, bazy danych oraz szczegóły implementacji ważniejszych klas.

Rozdział 1

Gry komputerowe

1.1 Wprowadzenie

Mianem gier komputerowych określa się aplikacje komputerowe, których przeznaczeniem jest dostarczanie rozrywki użytkownikowi. Należy zaznaczyć, że nie odnoszą się one jedynie do komputerów osobistych jak można sądzić po nazwie. W Polsce, a w zasadzie w języku polskim przyjęło się, że pojęcie komputer[1] jest równoznaczne z komputerem osobistym. Dlatego gra komputerowa zazwyczaj kojarzy się nam głównie z grą uruchamianą na komputerze osobistym. Jednak można nimi nazywać wszelkie gry uruchamiane na komputerach, tzn. również na konsolach do gier, telefonach komórkowych, palmtopach, automatach do gier itp.

Inną nazwą trochę rzadziej używaną od gier komputerowych są gry wideo. Pojęcie te jest zazwyczaj używane w przypadku gier konsolowych, gdy gra komputerowa jednoznacznie kojarzy się z grą na komputery osobiste. Jednak również ta nazwa nie odnosi się jedynie do gier na konsole (gier, które nie są uruchamiane na komputerach osobistych), ale również określa wszelkie gry uruchamiane na komputerach.

Natomiast grami elektronicznymi przyjęło się nazywać proste, małe i przenośne konsole do gier. Taka „gra elektroniczna" w dzisiejszych czasach jest urządzeniem niezwykle prostym w porównaniu do nowoczesnych przenośnych konsol do gier (pod względem architektury komputerowej). Jednak czynnikiem wyróżniającym „gry elektronicznej" jest jej bardzo niska cena. W takim urządzeniu zazwyczaj jest wgranych przynajmniej kilka prostych gier.

Podsumowując wszelkie aplikacje komputerowe umożliwiające interaktywną rozrywkę na komputerze (komputerze osobistym, konsoli do gier itp.) mogą być na-

[1]Komputer – urządzenie elektroniczne automatycznie przetwarzające informacje (dane) zapisane cyfrowo, służące do szybkiego wykonywania obliczeń, przechowywania, porządkowania i wyszukiwania danych oraz sterowania pracą innych urządzeń[1].

zywane zarówno grami komputerowymi bądź grami wideo. Nie należy nazywać ich grami elektronicznymi, gdyż nimi przyjęło się nazywać proste przenośne konsole do gier. W swojej pracy będę głównie używał nazwy gry komputerowe.

1.2 Gry strategiczne

Strategia jest grą, w której przy ustalaniu wyniku rozgrywki, duże znaczenie mają podejmowane przez graczy decyzje. W grach strategicznych położony jest nacisk na umiejętności planowania, dobierania odpowiednich strategii, taktyk, logicznego rozumowania. Gry strategiczne są zarówno grami planszowymi jak i gatunkiem gier komputerowych. Strategie komputerowe nazywa się po prostu komputerowymi grami strategicznymi.

Gry strategiczne można podzielić ze względu na sposób rozgrywki oraz podgatunki. Rozgrywka w grach strategicznych może być prowadzona w sposób turowy lub ciągły. Jeśli ruchy graczy są wykonywane na zmianę, to taką grę nazywa się grą turową. Przedstawicielami tego typu rozgrywki są między innymi *Civilization* i *Heroes of Might & Magic*. Natomiast gdy rozgrywka jest prowadzona w sposób ciągły, bez podziału na tury, wtedy mamy doczynienia z grą strategiczną czasu rzeczywistego. *Age of Empires* i *The Settlers* są przykładami serii gier, w których rozgrywka odbywa się bez podziału na tury.

Pierwsze komputerowe strategie dużo czerpały z planszowych odpowiedników. Rozgrywka zazwyczaj odbywała się turami. Były wydzielone fazy ruchu dla każdego z graczy. Wraz z czasem komputerowe gry strategiczne ewoluowały i oddalały się od pierwowzorów. Popularność zaczęły zdobywać gry czasu rzeczywistego, w których nie było podziału na tury.

Oprócz podziału ze względu na rozgrywkę, gry strategiczne można podzielić na podgatunki, tj. gry ekonomiczne, taktyczne czy 4X. W grach ekonomicznych najważniejszym czynnikiem jest ekonomia. W takich grach zazwyczaj kierujemy miastem bądź przedsiębiorstwem. Ważnym przedstawicielem tego gatunku jest seria *SimCity*, która oprócz tego jest zaliczana do gatunku symulatorów. Natomiast gry taktyczne są grami, w których kierujemy niewielką liczbą postaci. Naszym celem jest dobór odpowiedniej taktyki, aby pokonać przeciwników. W takich grach nie istnieje możliwość zbierania surowców. W grach 4X (eXplore, eXpand, eXploit, eXterminate) rozgrywka opiera się na 4 celach: eksploracji, ekspansji, eksploatacji i eksterminacji. Typowym przedstawicielem tego gatunku jest gra *Civilization*.

1.3 Menedżery piłkarskie

Do jakiego gatunku można przypisać grę typu menedżer piłkarski? Z pewnością jest strategią, gdyż wymaga strategicznego myślenia. W takiej grze gracz planuje każdy swój ruch. Ustalanie składu drużyny, odpowiedni dobór taktyki, formacji wymaga właśnie umiejętności planowania. Menedżery piłkarskie (sportowe) łączą w sobie elementy gier strategicznych, symulacyjnych i sportowych. Rozgrywka w takich grach polega na objęciu stanowiska menedżera klubu. Naszym celem jest wypracowanie pewnych strategii, które pozwolą na wygrywanie meczów, zdobywanie trofeów, utrzymanie równowagi finansowej. W menedżerze piłkarskim prowadzimy klub piłkarski przez cały sezon ligowy, który składa się z kolejek[2]. Pomiędzy nimi mamy nieograniczony czas na odpowiedni dobór strategii. Jako kierownik zespołu możemy ustalać skład, taktykę, treningi, dokonywać transferów, prowadzić finanse klubu.

1.3.1 Gry jednoosobowe

Football Manager

Football Manager jest serią gier komputerowych – menedżerów piłkarskich. Pierwsza odsłona pojawiła się w 1982 roku na komputer ZX Spectrum. Jej autorem był Kevin Toms. Gra okazała się dużym sukcesem. Football Manager był pierwszą grą takiego typu i zapoczątkował cały gatunek gier komputerowych – menedżerów piłkarskich[11]. W kolejnych latach gra została przepisana na wielu różnych platform, tj. Commodore 64, Amiga, PC. Football Manager został napisany w języku Basic. Gra przez większość czasu pracuje w trybie tekstowym. Jedynie w czasie meczów pojawiają się proste animacje przedstawiające sytuacje podbramkowe. Po rozpoczęciu gry pierwszym krokiem jest wybór drużyny. Do wyboru mamy kluby z ligi angielskiej. Grę rozpoczynamy w czwartej lidze. Naszym celem jest awans do najwyższego szczebla rozgrywek ligowych w Anglii. Oprócz pierwszej części Football Managera w kolejnych latach ukazały się następne odsłony. Ostatnią grą z serii był Football Manager 3 – pojawił się w 1992 roku.

W 2005 r. twórcy serii Championship Manager zrezygnowali ze współpracy z dotychczasowym wydawcą i przejeli prawa do nazwy Football Manager. Pierwszą częścią nowej serii był Football Manager 2005. Od tego czasu kolejne wydania po-

[2] *Kolejka* – (sport.) jeden z określonej liczby regularnie powtarzanych elementów składowych systemu rozgrywek sportowych[12]. Jeśli w lidze jest 16 drużyn, to aby wszystkie drużyny rozegrały mecze, musi być ich 8. Kolejką nazywa się właśnie te 8 meczów. W sezonie kolejek będzie 30, bo każda drużyna zagra z każdą dwukrotnie.

jawiają się co roku. W dzisiejszych czasach Football Manager jest jednym z najpopularniejszych przedstawicieli tego gatunku. Obecne wydania gry są bardzo rozbudowane. Pojawił się nawet prostszy tryb gry nie wymagający od gracza poświęcenia dużej ilości czasu na rozgrywkę. Football Manager posiada ogromną bazę danych klubów, piłkarzy, menedżerów i członków sztabu szkoleniowego. Gra wpływa nawet na decyzje podejmowane przez prawdziwe kluby piłkarskie. Przykładowo niektóre drużyny korzystają z bazy danych Football Managera, aby zdobywać informacje o przeciwnikach czy młodych talentach piłkarskich.

Championship Manager

Championship Manager jest jedną z najpopularniejszych serii menedżerów piłkarskich. Pierwsza część została wydana w roku 1992. Gra została napisana przez braci Collyer w ich domu w Shropshire w Anglii. Językiem programowania wykorzystanym do napisania gry był Basic. Championship Manager w przeciwieństwie do Football Managera nie zdobył dużej popularności już przy premierze pierwszej części. Dopiero wraz z kolejnymi odsłonami stał się bardzo popularnym przedstawicielem tego gatunku. W grze wydanej w 1992 roku mieliśmy do wyboru wszystkie drużyny ligi angielskiej z czterech najwyższych szczeblów rozgrywki. Przebieg meczu mogliśmy śledzić czytając komentarze pojawiające się na ekranie. Dużym mankamentem gry było brak prawdziwych nazwisk piłkarzy. O ile nazwy klubów odpowiadały rzeczywistym drużynom, to ich składy były już generowane losowo. W roku 1993 pojawiła się kolejna odsłona serii – Championship Manager '93. Gra została przepisana na język C, dodano prawdziwe nazwiska piłkarzy oraz wiele mniejszych nowości. Te wydanie gry sprzedawało się już znacznie lepiej od poprzedniej części. W następych latach pojawiały się aktualizacje sezonowe (zawierające wszystkie transfery dokonane przez ten czas) oraz całkiem nowe odsłony.

O'Leary Manager 2000

GameBoy Color jest przenośną konsolą do gier. Właśnie na tą konsolę w roku 2000 pojawiła się gra piłkarska O'Leary Manager 2000. Jest to połączenie gry zręcznościowej z piłkarską strategią. Do wyboru właściwie mamy dwa tryby. W jednym z nich możemy być menedżerem klubu – ustalać skład, taktykę, dokonywać transferów itp. Mecze w tym trybie są symulowane. Gracz nie ma bezpośrednio wpływu na wynik spotkania. Na rysunku 1.1 znajduje się widok rozgrywanego meczu.

Natomiast w drugim trybie oprócz wymienionych powyżej możliwości bierzemy czynny udział w meczu – gramy piłkarzami naszej drużyny. Gra pomimo dość słabej oprawy audio-wizualnej jest bardzo grywalna. Posiada intuicyjne menu oraz dość

Rysunek 1.1: Widok z gry *O'Leary Manager 2000*

mocno rozbudowany tryb menedżerski. W grze do wyboru mamy kluby piłkarskie z dwóch najwyższych szczeblów rozgrywek z ligi angielskiej, włoskiej, francuskiej, hiszpańskiej, holenderskiej i niemieckiej.

FIFA

FIFA to nie tylko skrót nazwy światowej federacji piłkarskiej, ale również seria gier komputerowych wydawana przez Electronic Arts (EA Sports). FIFA nie jest typowym menedżerem piłkarskim. Najogólniej mówiąc jest grą piłkarską. Najnowsze odsłony łączą w sobie elementy symulacji, gier zręcznościowych i strategicznych. Jednak tak nie było od początku.

Pierwszą częścią serii była FIFA International Soccer i pojawiła się w roku 1994. Była to gra typowo zręcznościowa – graliśmy piłkarzami na boisku. Do wyboru mieliśmy tylko drużyny narodowe. Z elementów pozaboiskowej rozgrywki mogliśmy zmieniać skład i taktykę drużyny. W kolejnych latach pojawiały się nowe odsłony, a w grze można było znaleźć coraz więcej elementów menedżera piłkarskiego.

Przełomową częścią serii była FIFA Football 2004. Po raz pierwszy umieszczono w niej pełnoprawny tryb kariery. Gracz mógł wcielić się w menedżera piłkarskiego i prowadzić swój klub. Od wydania tej odsłony można powiedzieć, że gra FIFA jest nie tyllko grą zręcznościowo – symulacyjną, ale również strategiczną.

1.3.2 Gry MMOG

Hattrick

Pod domeną *hattrick.org* znajduje się strona internetowa. Hattrick jest grą sieciową, w którą gramy za pomocą przeglądarki internetowej. Użytkownik nie musi instalować żadnego dodatkowego oprogramowania. Aby wziąć udział w grze należy

Rysunek 1.2: Strona internetowa *hattrick.org*

zarejestrować się na stronie. Hattrick należy do gier typu MMOG (Massively Multiplayer Online Game – Masowa Wieloosobowa Gra Internetowa). Oznacza to, że rozgrywka odbywa się za pomocą internetu. Każda osoba, która ma dostęp do sieci, może wziąć w niej udział, tzn. naszymi przeciwnikami są inni gracze z całego świata.

Hattrick jest menedżerem piłkarskim, w którym możemy wcielić się w prezesa klubu piłkarskiego. Wszystkie kluby w świecie Hattricka są fikcyjne – gracze sami wymyślają nazwy dla swoich drużyn. Rysunek 1.2 przedstawia widok tabeli ligowej w trakcie rozgrywanego sezonu piłkarskiego. Jak można zauważyć, drużyna znajduje się m. in. w VIII lidze, z czego jest 2048 lig na tym szczeblu rozgrywek.

Hattrick jest grą, w której aspekty strategiczne mają ogromne znaczenie. W prowadzeniu drużyny ważna jest również strategia długoterminowa – trenowanie piłkarzy czy prowadzenie akademii piłkarskiej. W przeciwieństwie do wielu innych gier MMOG zajmowanie się rozgrywką nie zabiera dużo czasu. W praktyce wystarczy zalogować się raz w tygodniu. Inną ważną częścią Hattricka jest społeczność. Podczas gry możemy korzystać z wielu forów i znajdować znajomych.

Rozdział 2

Tworzenie gier

Tworzenie komercyjnych gier w dzisiejszych czasach jest procesem długotrwałym, zajmującym często kilka lat. W przypadku gier niezależnych, zwanych grami *indie*, proces ten zazwyczaj jest znacznie krótszy. Często zdarza się, że gry indie są darmowe. W przypadku gdy tak nie jest, wtedy niezależne gry są dystrybuowane drogą elektroniczną z pominięciem wydawcy. W ostatnich latach branża gier niezależnych widocznie się rozrosła wraz z rozwojem internetu i rynku gier mobilnych.

Pierwsze gry powstały w latach pięćdziesiątych. Jednak komercyjne produkcje zaczęły powstawać dopiero w latach siedemdziesiątych. Wiązało się to z powstaniem pierwszej generacji konsol do gier oraz ze wzrostem rozpowszechnienia komputerów. Pierwsze komercyjne gry w porównaniu z obecnymi wymagały niewielkiego nakładu finansowego oraz ilości poświęconego czasu. Z tego powodu ilość osób zaangażowanych w produkcję gry była niewielka – zazwyczaj było to kilka osób. Dość często zdarzało się, że gry były tworzone przez pojedyncze osoby.

Wraz z rozwojem informatyki, gry stają się coraz bardziej rozbudowane, co powoduje znaczny wzrost nakładów finansowych i poświęconego czasu. Obecnie zdarza się, że do produkcji wielkich tytułów zaangażowanych jest ponad 1000 osób.

> *„Tak wygląda teraz nasza praca. Wszyscy pracują nad GTA czy RDR, a potem zajmujemy się nowym projektem. Obecnie potrzebujemy ponad 1000 osób, by stworzyć grę. To jest jedno z wymagań, jednak nie chcemy tylu osób w jednym miejscu."*[10]
>
> Leslie Benzies, prezes Rockstar North

Na drugim biegunie znajdują się gry indie. Są one tworzone przez małe zespoły lub przez pojedyncze osoby bez udziału deweloperów i wydawców. Takie gry mogą być tworzone od kilku dni, tygodni do nawet kilku lat w zależności od złożoności projektu i wielkości zespołu. Niezależne gry zazwyczaj są projektami znacznie

mniej rozbudowanymi od gier tworzonych przez profesjonalne studia. Brak wydawcy i dewelopera pozwala autorom na rozwijanie projektów oryginalnych. Autorzy niezależnych gier zwykle opierają się na dystrybucji cyfrowej przez internet. Dobrym przykładem gry niezależnej jest *Minecraft*. Początkowe wersje zostały napisane przez jedną osobę – Markusa Perssona.

2.1 Proces produkcji

Gry głównego nurtu zwykle są opracowywane w fazach. Najpierw jest tzw. *preprodukcja*. Gdy zostanie przygotowana już wstępna koncepcja, to następnie są projektowane elementy rozgrywki, pisana jest dokumentacja oraz tworzone są prototypy. Jeśli projekt zostanie zatwierdzony przez kierownictwo, wtedy zaczyna się etap *produkcji właściwej*.

W fazie produkcji właściwej zostają zaangażowane znaczne środki finansowe producenta oraz następuje konsolidacja pracowników studia. Właśnie w tej fazie produkcji gra jest rozwijana w pełnej skali – powstaje kod źródłowy, grafika, dźwięk, muzyka. Wielkość zespołu pracującego nad daną grą zależy od wielkości studia, złożoności gry oraz tempa produkcji. W popularnych i dużych produkcjach nad daną grą pracuje często ponad 100 osób. Każdy z pracowników odpowiada za inną część projektu. W zespole znajdują się tacy specjaliści jak: projektanci, graficy, muzycy, programiści, testerzy. Tworzenie gry, tak jak tworzenie każdej innej aplikacji, podlega cyklowi życia oprogramowania. Jest to seria kolejnych zmian w programie. W tym cyklu możemy wyróżnić kilka etapów – wersji gry. Pierwszą z nich jest *wersja robocza (pre-alpha)*, która dostępna jest tylko dla twórców programu. Tak nazywana jest wersja gry od początku pisania programu. Gdy zostaną zaimplementowane podstawowe funkcjonalności, gra przechodzi do *wersji alfa*. Jest to pierwsza wersja gry, która wychodzi poza grupę ludzi, przez którą została napisana. Zwykle w tym etapie gra zawiera wiele błędów. W wersji alfa gra jest testowana przez osoby spoza zespołu projektowego. Gdy większość gry została już ukończona, wtedy wydawawana jest *wersja beta*. W niej wszystkie funkcjonalności zostały zaimplementowane, a autorzy głównie skupiają się na usuwaniu błędów, które są znajdowane przez profesjonalnych testerów. Często wersje beta są również „uwalniane" do internetu przez producentów oprogramowania, aby chętni użytkownicy pomogli znajdować błędy. Ostatnią wersją przed oficjalnym wydaniem gry jest wersja *RC (ang. Release Candidate – Kandydat Do Wydania)*. Jest to etap produkcji, w którym gra jest uważana za skończoną i jedynie poważne błędy mogą zatrzymać wydanie gry. Jeśli program zostanie uznany za gotowy, wtedy wersja RC staje się ostateczną i nazywana jest *wersją stabilną*. W

tym momencie rozpoczyna się proces wydawania oprogramowania na rynek.

Po wydaniu gry rozpoczyna się etap *postprodukcji*, w którym następuje konserwacja oprogramowania. W przeszłości, głównie na konsolach starszych generacji, etap postprodukcji praktycznie nie istniał. Oprogramowanie było wypuszczane na rynek i na tym kończyła się praca producenta. Po prostu po wydaniu jednej gry zaczynał się kolejny proces produkcji nowej gry. Dopiero gdy znajdowano poważne błędy w wydanej grze, wtedy tworzono poprawione wersje oprogramowania. Jednak w dzisiejszych czasach, gdy dostęp do internetu jest powszechny, etap postprodukcji znacznie się przedłużył. Autorzy gier cały czas pracują nad poprawianiem znalezionych błędów, wydając kolejne *patch'e*. Dodatkowo gry są cały czas rozbudowywane, dodawane są nowe funkcjonalności. Dziś trudno jest całkowicie rozdzielić etap produkcji właściwej i postprodukcji, gdyż po wydaniu gry często jest dość intensywnie rozwijana przez autorów.

Oczywiście proces produkcji trochę inaczej wygląda w grach niezależnych tworzonych przez niewielką grupę osób bądź przez pojedynczego autora. Małe zespoły są bardziej elastyczne i mogą pozwolić sobie na pominięcie niektórych etapów produkcji. Zazwyczaj nie jest potrzebna rozbudowana dokumentacja, ani prototypy, gdyż członkowie zespołu doskonale zdają sobie sprawę jak gra ma wyglądać.

2.2 Języki programowania

Gdy zostaje uzgodniona wstępna koncepcja gry, nadchodzi moment wyboru języka programowania. Wybór ten zależy od wielu czynników, tj. powszechna znajomość języka wśród programistów czy platforma docelowa, na którą ma być wydana gra. Innym ważnym czynnikiem przy wyborze języka programowania jest również wybór bibliotek, z których zamierzamy korzystać. Wiele bibliotek jest przenośnych, tzn. dostępne są na różne systemy operacyjne. Często podejmowanie decyzji o wyborze języka idzie w parze z wyborem bibliotek.

W dzisiejszych czasach w programowaniu popularne są języki zorientowane obiektowo, tj. *C++, Java, C#*. Język C++ uznawany jest za najpopularniejszy język programowania do tworzenia gier wymagających dużych zasobów. Spowodowane jest to tym, że kod źródłowy napisany w C++ kompilowany jest bezpośrednio do kodu binarnego. Oprócz tego programista ma większe możliwości pisania aplikacji w sposób niskopoziomowy niż w językach tj. Java i C#.

W języku Java kod jest kompilowany do kodu pośredniego, który jest wykonywany przez wirtualną maszynę. Dzięki temu powstała aplikacja jest niezależna od architektury komputera i systemu operacyjnego. Jednak jest to obarczone kosztem

wydajnościowym. Aplikacje, które są kompilowane do kodu pośredniego są mniej wydajne od aplikacji kompilowanych do kodu binarnego. Dlatego język Java bardziej nadaje się do pisania prostych gier mobilnych dostępnych na wiele różnych systemów niż do pisania skomplikowanych gier wymagających dużej szybkości działania. Z tego powodu Java jest popularna w prostych grach wydawanych na urządzenia mobilne.

2.3 Biblioteki

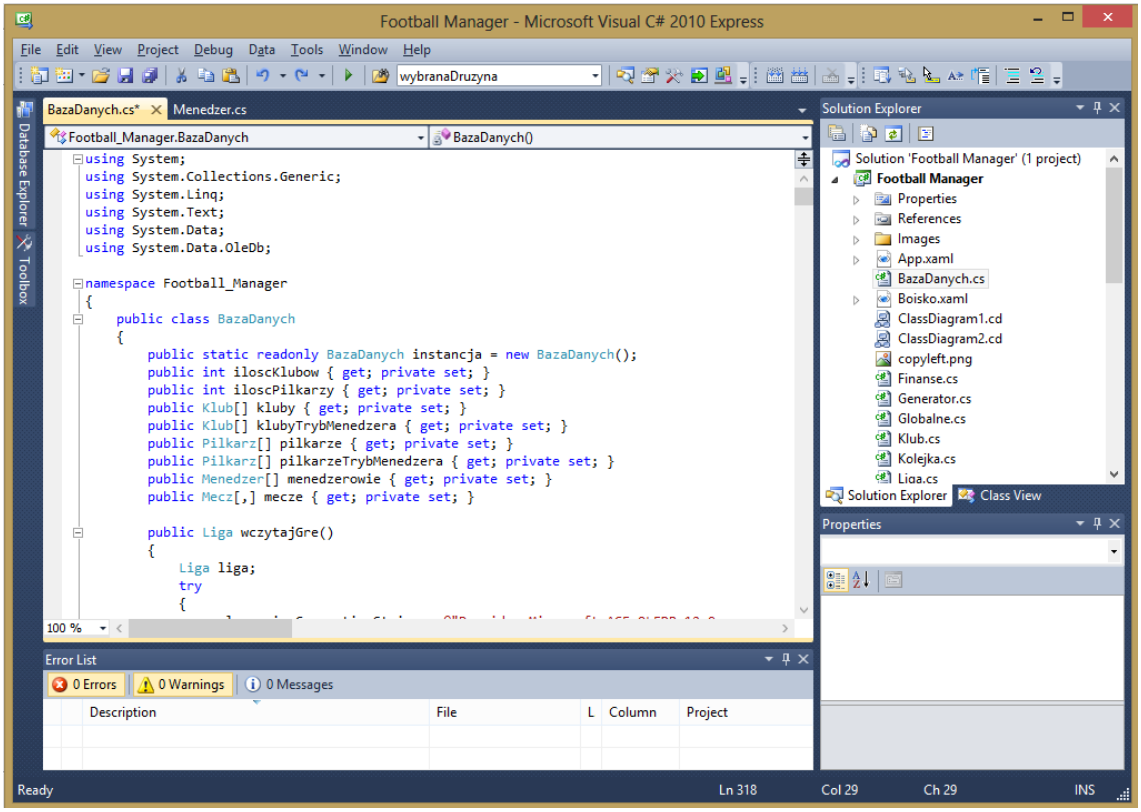
Bardzo ważną decyzją jest wybór bibliotek, z których zamierzamy korzystać przy programowaniu aplikacji. W dzisiejszych czasach istnieje bardzo wiele bibiliotek, które wspomagają proces programowania. Niektóre biblioteki jedynie obsługują wyświetlanie grafiki czy sterowanie dźwiękiem. Natomiast istnieją również biblioteki bardziej rozbudowane, które wspomagają praktycznie cały proces programowania gier. Oczywiście biblioteki można podzielić również pod względem grupy docelowej – jedne są bardzo łatwe w nauce oraz są przeznaczone raczej do prostszych produkcji. Przykładami takich bibliotek są np. *Allegro* oraz *SDL*. Natomiast inne dla nowicjuszy mogą być trudne w programowaniu, ale za to posiadają ogromne możliwości i często są wykorzystywane w produkcjach komercyjnych. *DirectX* oraz *OpenGL* są przykładami takich bibliotek.

Wybór danych bibliotek zależy od platformy, na którą ma zostać wydana gra. Wiele bibliotek jest przenośnych (Allegro, OpenGL, SDL) na różne systemy operacyjne. Jednak istnieją również biblioteki, które nie są przenośne. DirectX jest przykładem takiej biblioteki. DirectX jest dostępny jedynie na „produkty" firmy Microsoft: system operacyjny Windows oraz konsolę Xbox.

Obecnie większość komercyjnych produkcji na system operacyjny Windows wykorzystuje bibliotekę DirectX. Jeśli wydawca gry chciałby wydać grę np. na system Linux, musiałby przepisać grę na inną bibliotekę. Przykładowo gra *SimCity* na systemie operacyjnym Windows korzysta z DirectX. Natomiast ta sama gra wydana pod system Linux korzysta już z biblioteki SDL.

2.4 Środowiska programistyczne

Gry komputerowe tak jak inne programy są generowane z kodu źródłowego do pliku wykonywalnego przez kompilator. Kod źródłowy może być pisany w wielu edytorach tekstu, np. w Notatniku w systemie Windows. Następnie kod źródłowy należy skompilować za pomocą specjalnego programu – kompilatora, który tłumaczy



Rysunek 2.1: Screen z IDE *Microsoft Visual C# 2010 Express*

kod napisany przez programistę na kod maszynowy zrozumiały dla komputera.

Jednak aby uprościć i przyspieszyć tworzenie oprogramowania powstają tzw. *IDE (Integrated Development Environment – Zintegrowane Środowiska Programistyczne)*. IDE są aplikacjami, która dostarczają programistom kompleksowe narzędzia do tworzenia oprogramowania. Zintegrowane środowiska programistyczne zazwyczaj zawierają edytor kodu źródłowego, kompilator, debugger oraz inne narzędzia służące do automatyzacji tworzenia kodu. Oczywiście IDE nie jest konieczne, aby tworzyć oprogramowanie, ale pozwala ten proces znacznie uprościć i przyspieszyć.

Często zintegrowane środowiska programistyczne pozwalają na tworzenie aplikacji w różnych językach. Jednak zazwyczaj IDE są tworzone pod konkretne języki programowania. Przykładowo *Code::Blocks* oraz *Dev-C++* zostały napisane z myślą o języku C/C++. Z kolei *Eclipse* oraz *NetBeans* głównie są wykorzystywane przy pisaniu aplikacji w języku Java. Microsoft posiada cały „zestaw" zintegrowanych środowisk programistycznych pod nazwą *Visual Studio*. Ułatwiają one również pisanie aplikacji graficznych, dzięki możliwości tworzenia interfejsu graficznego aplikacji za pomocą kontrolek. Na rysunku 2.1 znajduje się screen ze środowiska programistycznego Visual C# 2010 Express.

2.5 Inne narzędzia

Korzystanie z gotowych rozwiązań znajdujących się w bibliotekach oraz tworzenie kodu w zintegrowanych środowiskach programistycznych znacznie ułatawiają tworzenie oprogramowania. Oczywiście poza tymi narzędziami są inne, które wspomagają proces programowania aplikacji. Przy tworzeniu gier, w których przedstawiony świat jest trójwymiarowy, wykonywane są różne modele postaci, przedmiotów i całego otoczenia. Właśnie przy pracach nad takimi rzeczami wykorzystywane jest specjalne oprogramowanie do tworzenia trójwymiarowych modeli. Przykładami programów do grafiki 3d są m. in. *Blender* oraz *3ds Max*.

Blender jest profesjonalnym i darmowym programem, wydawanym na licencji GNU GPL (General Public License). Program jest rozwijany przez Blender Foundation. Dostępny jest na wiele systemów operacyjnych, tj. Windows, Linux, OS X. Wbrew pozorom, które można odnieść przez to, że Blender jest darmowym oprogramowaniem, jest to profesjonalne środowisko używane przez ludzi pracujących w różnych branżach. Podstawową funkcjonalnością programu jest modelowanie obiektów. Mogą one zostać wykorzystane np. w grach. Z prostych obiektów można stworzyć otoczenie, które przypomina świat rzeczywisty. Takie otoczenie może być wykorzystane do renderowania obrazów bądź animacji. Blender nadaje się do tworzenia filmów animowanych, co udowodnili m.in. twórcy filmu *Sintel*[7]. Trójwymiarowe modele, które zostały stworzone za pomocą Blendera nie muszą pozostać w świecie wirtualnym. Program pozwala na wydrukowanie ich za pomocą drukarek 3d[6].

3ds Max jest to profesjonalny i komercyjny program służący do tworzenia trójwymiarowych modeli, obrazów, animacji i gier. Wydawany jest na system operacyjny Microsoft Windows przez firmę Autodesk. 3ds Max jest bardzo rozbudowanym programem, pozwalającym na obsługę wielu rozszerzeń. Posiada również własny język skryptowy. Najpopularniejszymi zastosowaniami programu są tworzenie gier komputerowych oraz wizualizacje świata rzeczywistego. 3ds Max często jest wykorzystywany przy produkcji filmów animowanych jak i efektów specjalnych w zwykłych filmach. Przykładowo krótkometrażowy film *Katedra* Tomasza Bagińskiego powstał w 3ds Max[9]. Oprócz tego oprogramowanie jest również wykorzystywane do wizualizacji architektonicznych.

Rozdział 3

Aplikacja

3.1 Koncepcja gry

Gra jest przedstawicielem gatunku menedżerów piłkarskich. Jest to typ gier łączący w sobie elementy symulacji, gier strategicznych i sportowych. Gra jest aplikacją okienkową. Interfejs użytkownika (przyciski, tabele) są typowe dla aplikacji pracujących w systemie Windows.

W grze wcielamy się w rolę menedżera klubu. Stajemy się zarówno kierownikiem jak i trenerem drużyny. Do wyboru mamy dwa tryby gry: możemy rozegrać mecz towarzyski lub rozpocząć tryb menedżera. W meczu towarzyskim rozgrywamy pojedyncze spotkanie wybranymi przez nas drużynami. Przed rozpoczęciem meczu możemy przeglądać piłkarzy, ich umiejętności oraz ustalić wyjściową jedenastkę piłkarzy. W oknie taktyka ustalamy formację, ustawienie, pressing, poziom agresji, kapitana drużyny czy wykonawcę stałych fragmentów gry.

Natomiast w trybie menedżera wybieramy zespół, który chcemy poprowadzić przez sezon ligowy. W lidze gra 16 drużyn. Jest 30 kolejek po 8 meczów, co daje 240 spotkań w całym sezonie. Pomiędzy kolejkami możemy dokonywać zmian w składzie, formacji, ustawienia drużyny. Celem gracza jest wygrywanie meczów i zajęcie jak najlepszej pozycji w lidze.

3.2 Wykorzystane technologie informatyczne

Językiem programowania wykorzystanym do napisania aplikacji jest *C#*. Jest to język o uniwersalnym przeznaczeniu. Dobrze nadaje się do pisania aplikacji okienkowych pod system Windows. Narzuca pisanie programów w paradygmacie obiektowym, który jest standardem w obecnych czasach – zwłaszcza przy pisaniu aplikacji w trybie graficznym oraz gier komputerowych. O takim wyborze zadecydowało to,

że wywodzi się z języka *C++*, od którego według mnie różni się tym, że posiada wiele ciekawych rozwiązań i udogodnień, które pozwalają programiście bardziej skupić się na części logicznej aplikacji. Język C# posiada m. in. mechanizm odzyskiwania pamięci (Garbage Collector), który uwalnia programistę od problemów z zarządzaniem pamięcią, tj. przecieki pamięci [3, str. 20]. Wraz z wyborem języka C# zostały dwa interfejsy służące do programowania aplikacji graficznych w systemie Windows:

- *Windows Forms*
- *Windows Presentation Foundation*

Do utworzenia interfejsu graficznego zostało użyte *Windows Presentation Foundation*. Jest to ciekawa technologia firmy Microsoft, która pozwala oddzielić część wizualną aplikacji od części logicznej. Część wizualną piszemy za pomocą języka *XAML (eXtensible Application Markup Language)*, który jest „odmianą" *XML* stworzoną przez Microsoft. W XAML skupiamy się jedynie na wyglądzie aplikacji. Używamy do tego znaczników, które znamy z języka HTML. Natomiast cała część logiczna aplikacji jest już pisana za pomocą zwykłego kodu w oddzielnym pliku. Oprócz tego *WPF* korzysta z DirectX do renderowania grafiki [2, str. 19]. Pozwala to na wykorzystanie karty graficznej przy działaniu aplikacji napisanej w WPF.

Wybranym środowiskiem programistycznym jest *Visual C# 2010 Express*. Jest to wersja darmowa, zawierająca wszystkie niezbędne narzędzia do tworzenia aplikacji komputerowych. Na taki wybór wpłynęło to, że Visual C# wspiera wymieniony już wyżej interfejs – Windows Presentation Foundation.

Aplikacja korzysta z relacyjnej bazy danych za pomocą języka *SQL*. O takim wyborze zadecydowało to, że dobrze jest przedstawiać dane z paradygmatu obiektowego w postaci relacyjnej bazy danych. Do stworzenia bazy został wykorzystany *Microsoft Access 2010*. Aplikacja łączy się z bazą danych za pomocą interfejsu *OleDB*.

3.3 Struktura aplikacji – spis klas

Gra została napisana w obiektowym paradygmacie programowania. W aplikacji jest 11 klas odpowiadających tylko za część logiczną gry. Dodatkowo do napisania interfejsu graficznego zostało utworzonych 15 *kontrolek*[1] użytkownika (*User Control*), z czego każda kontrolka jest również oddzielną klasą.

[1]Kontrolki (obiekty kontrolne) są to elementy interfejsu tj. przyciski, listy wyboru, pola wyboru czy pola tekstowe wykorzystywane w aplikacjach pracujących na danym systemie operacyjnym. Wiele obiektów kontrolnych jest wbudowanych w system operacyjny.

3.3.1 Część logiczna

- *Akcja* – Jest przedstawieniem jednej akcji meczowej. Akcji jest 90 – tyle ile minut trwa mecz piłkarski. Zawiera jedynie trzy zmienne: *minuta, komentarz* oraz *pilkePosiadaGospodarz.*

- *BazaDanych* – Służy do wczytywania danych z bazy. Istnieje jedna instancja tej klasy i dostęp do niej jest globalny. W tej klasie tworzone są tablice obiektów klubów, piłkarzy, menedżerów.

- *Generator* – Klasa służąca do generowania danych, tj. kluby, menedżerowie, piłkarze i umieszczania ich w relacyjnej bazie danych.

- *Klub* – Zawiera wszystkie informacje o klubie, w tym listę piłkarzy (tablica obiektów *Pilkarz*), taktykę, statystyki klubu. Oprócz tego, w tej klasie znajdują się różne metody, m. in. odpowiadające za obliczanie poziomów drużyn.

- *Kolejka* – Jest przedstawieniem kolejki ligowej. Zawiera konstruktor przypisujący mecze danym kolejkom, tak aby wszystkie drużyny rozegrały mecze każdy z każdym dwukrotnie.

- *Liga* – Odpowiada za obsługę rozgrywania sezonu ligowego. W tej klasie znajdują się m.in. tabela, kluby, kolejki oraz metody *rozegrajKolejke*, *sprawdzMiejsce* czy *uaktualnijStatystyki.*

- *Mecz* – Klasa, która odpowiada za symulację meczu. Składa się m. in. z 90 obiektów klasy *Akcja* oraz listy obiektów *Zdarzenie.*

- *Menedżer* – Zawiera dane osobowe na temat menedżera. W tej klasie znajdują się metody odpowiedzialne za ustalanie optymalnych składów drużyn.

- *Pilkarz* – Klasa będąca przedstawieniem piłkarza. Zawiera jego dane osobowe, przynależność do klubu, poziomy umiejętności.

- *Tabela* – Jest wykorzystywana do przedstawienia tabeli ligowej. Zawiera tablicę obiektów *Klub* oraz metody odpowiedzialne m. in. za sortowanie.

- *Zdarzenie* – Klasa przedstawiająca zdarzenie w meczu piłkarskim. Takim zdarzeniem jest zdobycie gola czy podyktowanie rzutu karnego. Przykładowo, gdy jedna z drużyn zdobędzie gola, tworzony jest obiekt tej klasy wraz z podaną minutą meczu, strzelcem gola oraz informacją, że zdarzeniem jest zdobyta bramka.

3.3.2 Interfejs graficzny i obsługa zdarzeń

MainWindow jest klasą dziedziczącą po *Windows*, co oznacza, że jest to kontrolka – okno. Reszta klas, oprócz *App*, dziedziczy po *UserControl*, tzn. są kontrolkami użytkownika. *MainWindow* pełni rolę swoistego pojemnika na obiekty *UserControl*. W *MainWindow* są umieszczone kontrolki, których nazwa zaczyna się od *Okno*. Wszystkie one oprócz *OknoGlowne* mają ustalną widoczność na ukrytą. W przypadku przechodzenia pomiędzy podstronami aplikacji, widoczności poszczególnych kontrolek zmieniają się.

- *App*
- *Boisko*
- *MainWindow*
- *OknoDruzynyTrybMenedzera*
- *OknoGlowne*
- *OknoMecz*
- *OknoMeczTowarzyski*
- *OknoOGrze*
- *OknoPoMeczu*
- *OknoPrzedMeczem*
- *OknoSklad*
- *OknoTabela*
- *OknoTaktyka*
- *OknoTerminarz*
- *OknoTerminarzDruzyny*
- *OknoTrybMenedzera*

3.4 Baza danych

Aplikacja korzysta z relacyjnej bazy danych za pomocą języka *SQL*. Baza danych została stworzona w programie *Microsoft Access 2010*. W aplikacji znajdują się dwie klasy łączące się z bazą danych. Są to: *Generator* i *BazaDanych*. Obydwie łączą się z bazą za pomocą interfejsu *OleDB*.

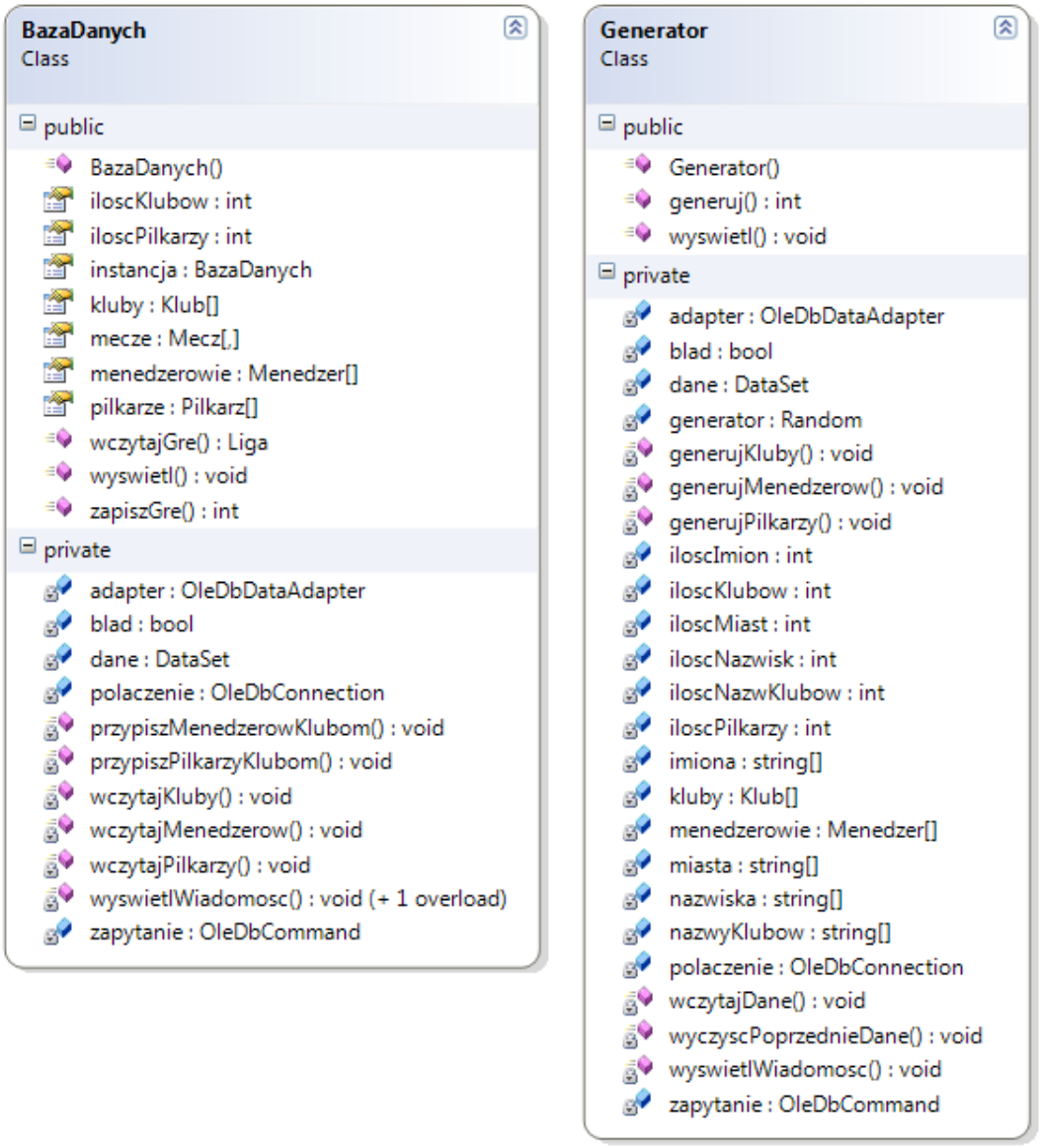
3.4.1 Struktura relacji

Baza danych składa się z dwóch plików:

- *generator* – tabele służące do generowania danych:
  - *Imie* – spis imion służący do generowania danych osobowych piłkarzy i menedżerów;
  - *Nazwisko* – spis nazwisk służący do generowania danych osobowych piłkarzy i menedżerów;
  - *NazwaKlubu* – spis nazw klubów służący do generowania pełnych nazw klubów składających się z nazwy klubu i miasta;
  - *Miasto* – spis miast wykorzystywanych do generowania drugiej części nazwy klubów;
- *baza danych* – tabele przechowujące już wygenerowane dane wykorzystywane bezpośrednio w grze:
  - *Klub* – spis wszystkich klubów występujących w rozgrywkach;
  - *Pilkarz* – spis wszystkich piłkarzy występujących w rozgrywkach wraz z przypisaniem do klubu;
  - *Menedżer* – spis wszystkich menedżerów prowadzących kluby;
  - *StanGry* – w tej tabeli przechowywane są szczegóły dotyczace zapisu gry;
  - *Mecz* – lista zapisanych meczów.

3.4.2 Generowanie losowych danych – klasa *Generator*

Za obsługę generowania danych odpowiada klasa *Generator*. W tabelach *Imie, Nazwisko, NazwaKlubu, Miasto* w bazie *generator* znajdują się wpisane dane, na podstawie których aplikacja generuje piłkarzy, menedżerów i kluby. Jest około 80 imion, ponad 900 nazwisk, ponad 60 nazw klubów oraz 30 miast. Proces generowania danych odbywa się etapami.



Rysunek 3.1: Diagramy klas *BazaDanych, Generator*

Najpierw aplikacja łączy się z bazą *generator*. Zostają wczytane imiona, nazwiska, nazwy klubów, miasta. Następnie, w czasie działania aplikacji, rekordy poszczególnych tabel są dobierane losowo. Na podstawie tych danych tworzone są nowe elementy – kluby bądź osoby. Zostają one umieszczone w pliku *baza danych*.

Przykładowo spośród wszystkich imion i nazwisk dla każdego piłkarza zostanie wylosowane imię i nazwisko. Oprócz tego losowane będą jego umiejętności, rok urodzenia. Zgodnie z tym schematem zostanie wylosowanych 320 piłkarzy, 16 klubów i 16 menedżerów. Aby zapisać wygenerowane dane aplikacja połączy się z plikiem *baza danych* i utworzy rekordy odpowiadające piłkarzom, klubom, menedżerom.

3.4.3 Wczytywanie zapisanych danych – klasa *BazaDanych*

Do wczytywania danych wykorzystywana jest klasa *BazaDanych*. W niej znajdują się funkcje opowiadające za pobieranie rekordów z bazy. Na listingu 3.1 przedstawiona jest metoda *wczytajPilkarzy*. Poniżej znajduje się zwięzły opis tego, co się dzieje w konstruktorze klasy znajdującego się na listingu 3.2. Najpierw klasa łączy się z bazą znajdującą się w pliku *baza danych*. Kolejno są wczytywane kluby, piłkarze i menedżerowie. Następnie klubom przypisywani są menedżerowie i na odwrót – menedżerom przypisywane są kluby. Mówiąc bardziej „programistycznie" – w obiekcie klasy *Klub* znajduje się referencja do obiektu klasy *Menedzer*. Podobnie klubom przypisywani są piłkarze. Z tą różnicą, że *Klub* zawiera tablicę 20 obiektów klasy *Pilkarz*. Na sam koniec poszczególnym piłkarze są sortowani według numerów, obliczane są poziomy klubów i uaktalniane wyjściowe jedenastki.

Listing 3.1: Metoda *wczytajPilkarzy* klasy *BazaDanych*

```
void wczytajPilkarzy()
{
  try
  {
    polaczenie.Open();
    zapytanie = new OleDbCommand("SELECT * FROM Pilkarz ORDER BY Nazwisko",
      polaczenie);
    adapter = new OleDbDataAdapter(zapytanie);
    dane = new DataSet();
    adapter.Fill(dane, "Dane");
    for (int i = 0; i < iloscPilkarzy; i++)
    {
      pilkarze[i] = new Pilkarz( ... );
    }
    polaczenie.Close();
  }
  catch (OleDbException e)
  {
    polaczenie.Close();
    wyswietlWiadomosc(e.Message);
  }
}
```

Listing 3.2: Konstruktor klasy *BazaDanych*

```
BazaDanych()
{
  try
  {
    polaczenie = new OleDbConnection();
    polaczenie.ConnectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;
                                    Data Source=..\..\..\baza danych.accdb;";
    polaczenie.Open();
    zapytanie = new OleDbCommand("SELECT COUNT(*) FROM Klub", polaczenie);
    iloscKlubow = (int)zapytanie.ExecuteScalar();
    zapytanie = new OleDbCommand("SELECT COUNT(*) FROM Pilkarz", polaczenie);
    iloscPilkarzy = (int)zapytanie.ExecuteScalar();
    kluby = new Klub[iloscKlubow];
    menedzerowie = new Menedzer[iloscKlubow];
    pilkarze = new Pilkarz[iloscPilkarzy];
    polaczenie.Close();
  }
  catch (Exception e)
  {
    polaczenie.Close();
    wyswietlWiadomosc(e.Message);
    blad = true;
  }
  if (blad == false)
  {
    wczytajKluby();
    wczytajPilkarzy();
    wczytajMenedzerow();
    przypiszMenedzerowKlubom();
    przypiszPilkarzyKlubom();
    for (int i = 0; i < iloscKlubow; i++)
    {
        kluby[i].sortujPilkarzWedlugNumerow();
        kluby[i].obliczPoziomy();
        kluby[i].uaktualnijWyjsciowa11();
    }
  }
}
```

3.5 *Windows Presentation Foundation*

3.5.1 Część wizualna – tworzenie interfejsu użytkownika

Gra jest aplikacją okienkową. Do utworzenia części wizualnej i obsługi zdarzeń została wykorzystana technologia *Windows Presentation Foundation (WPF)*. Jest to interfejs służący do pisania aplikacji graficznych pod system Windows. Tym czym się wyróżnia WPF od innych interfejsów API jest rozdzielenie części wizualnej od logicznej. Część wizualną aplikacji piszemy w dokumencie *XAML – eXtensible Application Markup Language*. Jest to język opisu interfejsu użytkownika. XAML opiera swoją składnię na XML. Pisanie części wizualnej aplikacji trochę przypomina pisanie dokumentu HTML strony internetowej. Pliki części wizualnej mają rozszerzenie xaml. Natomiast część logiczną piszemy zwykłym kodem w języku C# w plikach z rozszerzeniem xaml.cs.

W tworzeniu aplikacji szeroko zostały wykorzystane *User Control* – kontrolki tworzone przez użytkownika. User Control pozwala na stworzenie kontrolki składającej się z wielu zwykłych kontrolek. W aplikacji User Control zostało użyte do stworzenia sieci podstron. Program składa się z *MainWindow* – okna aplikacji. W *MainWindow* znajdują się wszystkie kontrolki wykorzystywane jako podstrony. Fragment kodu XAML klasy *MainWindow* znajduje się na listingu 3.3. Wszystkie podstrony oprócz User Control *OknoGlowne* mają ustawioną widoczność na ukrytą.

3.5.2 Część logiczna – obsługa zdarzeń

W Windows Presentation Foundation część logiczna aplikacji została widocznie rozdzielona od części wizualnej. Aby dodać obsługę zdarzeń dla kontrolki najpierw podajemy jej nazwę, a potem dopisujemy typ zdarzenia wraz z jego nazwą.

Na listingu 3.4, mamy definicję przycisku. Posiada nazwę *OGrze* oraz deklarację obsługi zdarzenia *Click* (wciśnięcie przycisku) o nazwie *OGrze_Click*. Z kolei na listnigu 3.5 jest fragment kodu z pliku xaml.cs. Szkielet funkcji jest tworzony automatycznie, gdy zadeklarujemy zdarzenie w pliku xaml. W tej funkcji ukrywamy obecne okno oraz zmieniamy stan *OknoOGrze* na widoczne. Obsługa okienek, tzn. zmiana ich widoczności w całej aplikacji została napisana w podobny sposób.

Listing 3.3: Kod *XAML* klasy *MainWindow*

```
<Window x:Class="Football_Manager.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:ucl="clr-namespace:Football_Manager"
    Title="Menedżer piłkarski"
    Height="600" Width="800"
    ResizeMode="CanMinimize">
  <Grid ShowGridLines="False" Margin="5">
    <Grid.RowDefinitions>
      <RowDefinition Height="135"/>
      <RowDefinition/>
    </Grid.RowDefinitions>
    <Image Grid.Row="0" Margin="0 10 0 0" VerticalAlignment="Center"
      Source="logo.png"/>
    <Grid Grid.Row="1" ShowGridLines="False" Margin="5">
      <ucl:OknoGlowne x:Name="oknoGlowne" Visibility="Visible"/>
      <ucl:OknoMeczTowarzyski x:Name="oknoMeczTowarzyski" Visibility="Hidden"/>
      <ucl:OknoDruzynyTrybMenedzera x:Name="oknoDruzynyTrybMenedzera"
           Visibility="Hidden"/>
      <ucl:OknoOGrze x:Name="oknoOGrze" Visibility="Hidden"/>
      <ucl:OknoTrybMenedzera x:Name="oknoTrybMenedzera" Visibility="Hidden"/>
      <ucl:OknoSklad x:Name="oknoSklad" Visibility="Hidden"/>
      <ucl:OknoTaktyka x:Name="oknoTaktyka" Visibility="Hidden"/>
      ...
    </Grid>
  </Grid>
</Window>
```

Listing 3.4: Opis przycisku za pomocą kodu *XAML*

```
<Button Grid.Row="4" Margin="10" Name="OGrze" Click="OGrze_Click">O grze</Button>
```

Listing 3.5: Metoda obsługująca kliknięcie przycisku

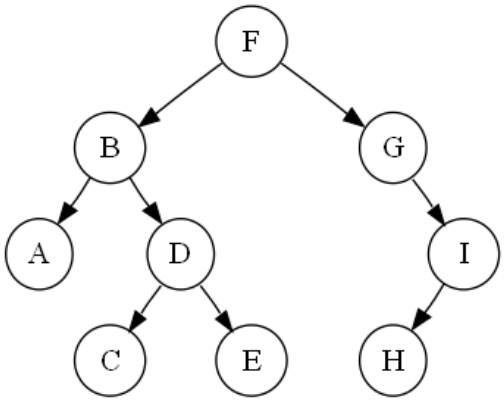
```
private void OGrze_Click(object sender, RoutedEventArgs e)
{
     this.Visibility = Visibility.Hidden;
     OknoOGrze.instancja.Visibility = Visibility.Visible;
}
```

3.6 Drzewa decyzyjne

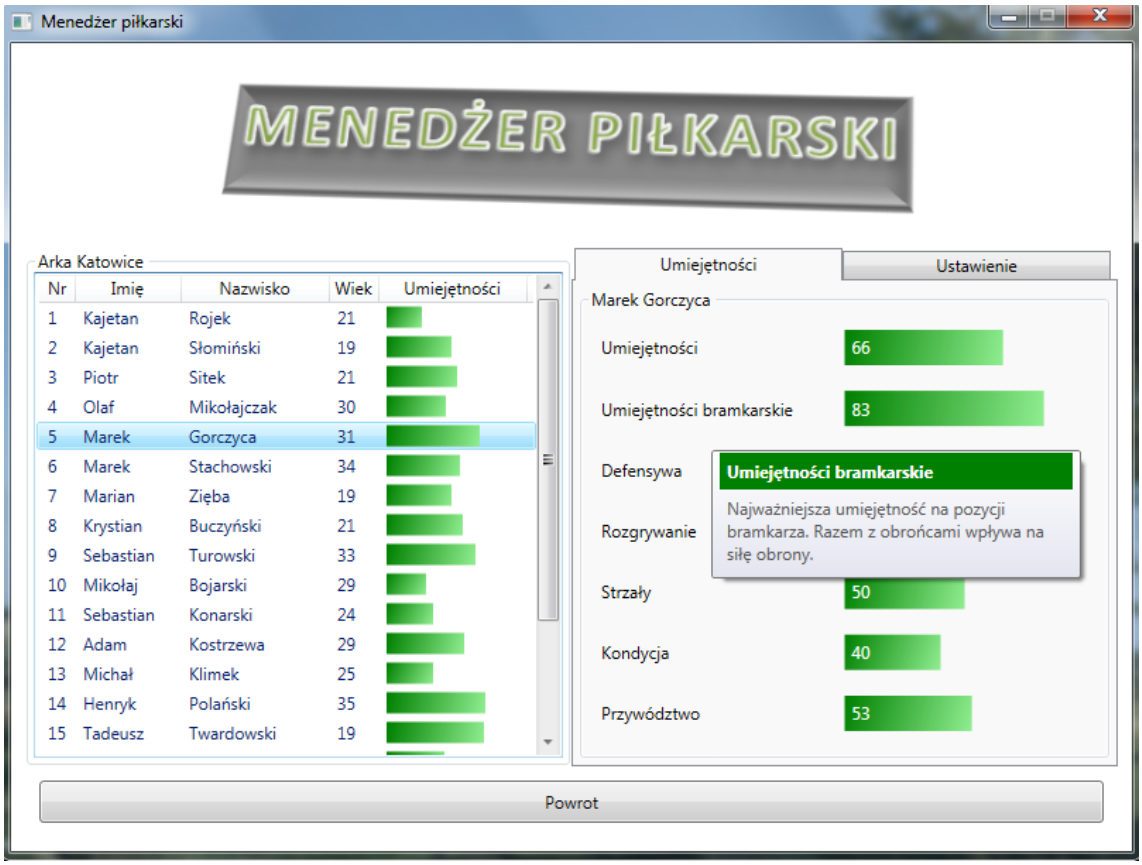
Definicja 1 *Drzewem* nazywamy graf spójny bez cykli, to znaczy taki graf, w którym dla każdej pary wierzchołków $u, v \in V(G)$ istnieje ścieżka o końcach w u i v, ale żadna ścieżka długości co najmniej dwa nie jest zamknięta krawędzią łączącą jej końce. Drzewo na n wierzchołkach ma oczywiście $n-1$ krawędzi. Wynika stąd, że suma stopni wierzchołków drzewa wynosi $2(n-1)$. Ponadto drzewo posiada co najmniej dwa wierzchołki stopnia 1 (zwane *wierzchołkami wiszącymi*)[5].

Zatem, graf jest drzewem, jeżeli jest spójny i acykliczny. Graf jest spójny wtedy, gdy pomiędzy każdymi dwoma wierzchołkami istnieje ścieżka je łącząca. Drzewo musi być acykliczne, tzn. nie można z żadnego wierzchołka przejść grafu i wrócić do niego samego. Liśćmi drzewa nazywa się węzły (wierzchołki), z których nie wywodzą się inne krawędzie. Natomiast węzły wewnętrzne są to wierzchołki, które nie są końcowe, tzn. wywodzą się z nich inne krawędzie. Na rysunku 3.2 znajduje się przykład drzewa binarnego.

Drzewami decyzyjnymi nazywa się drzewa służące do graficznego przedstawienia procesu decyzyjnego. Są szczególnie przydatne przy podejmowaniu decyzji, gdy do wyboru mamy wiele rozgałęziających się wariantów. Drzewo decyzyjne zawiera etykiety. Węzły wewnętrzne odpowiadają testom na wartościach atrybutów. Drzewo decyzyjne w danym węźle posiada tyle rozgałęzień, ile jest możliwych wyników testu. Liście są wynikami tych testów i zawierają decyzję o klasyfikacji przykładów. Tworząc program na podstawie drzewa decyzyjnego w praktyce programujemy same instrukcje warunkowe.



Rysunek 3.2: Drzewo binarne



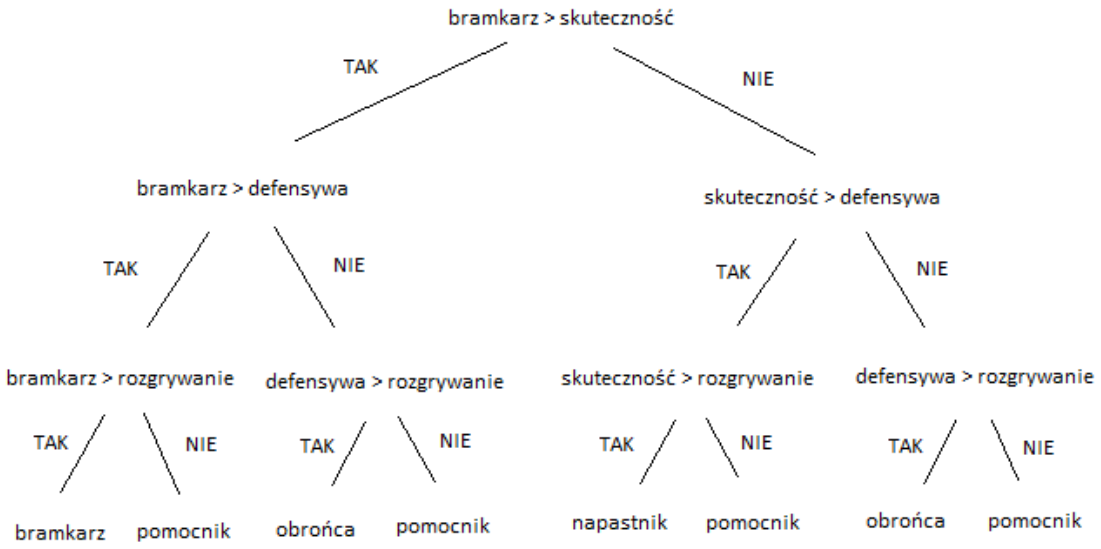
Rysunek 3.3: Screen z aplikacji przedstawiający skład drużyny

3.7 Ustalanie składów

Gdy rozgrywamy mecz towarzyski, możemy ustalać skład naszej drużyny (rysunek 3.3). Natomiast ustawienie pierwszej jedenastki piłkarzy przeciwnika jest ustalane przez komputer. Podobnie jest w trybie menedżera, w którym kierujemy jednym z 16 zespołów. W przypadku pozostałych 15 drużyn, ich optymalne składy również są ustalane przez komputer. Za wybór składów drużyn przeciwników odpowiada klasa *Menedzer*. Znajdują się w niej m. in. metody:

- *ustalNajlepszePozycjeDlaPilkarzy*
- *obliczIluJestPilkarzyNaDanePozycje*
- *sortuj*
- *zbilansujPozycjePilkarzy*
- *ustalNajlepszySklad*

W pierwszej kolejności ustalane są najlepsze pozycje (bramkarz, obrońca, pomocnik, napastnik) dla zawodników. Do ustawienia składu zostało zastosowane drzewo



Rysunek 3.4: Drzewo decyzyjne – znajdowanie najlepszej pozycji dla piłkarzy

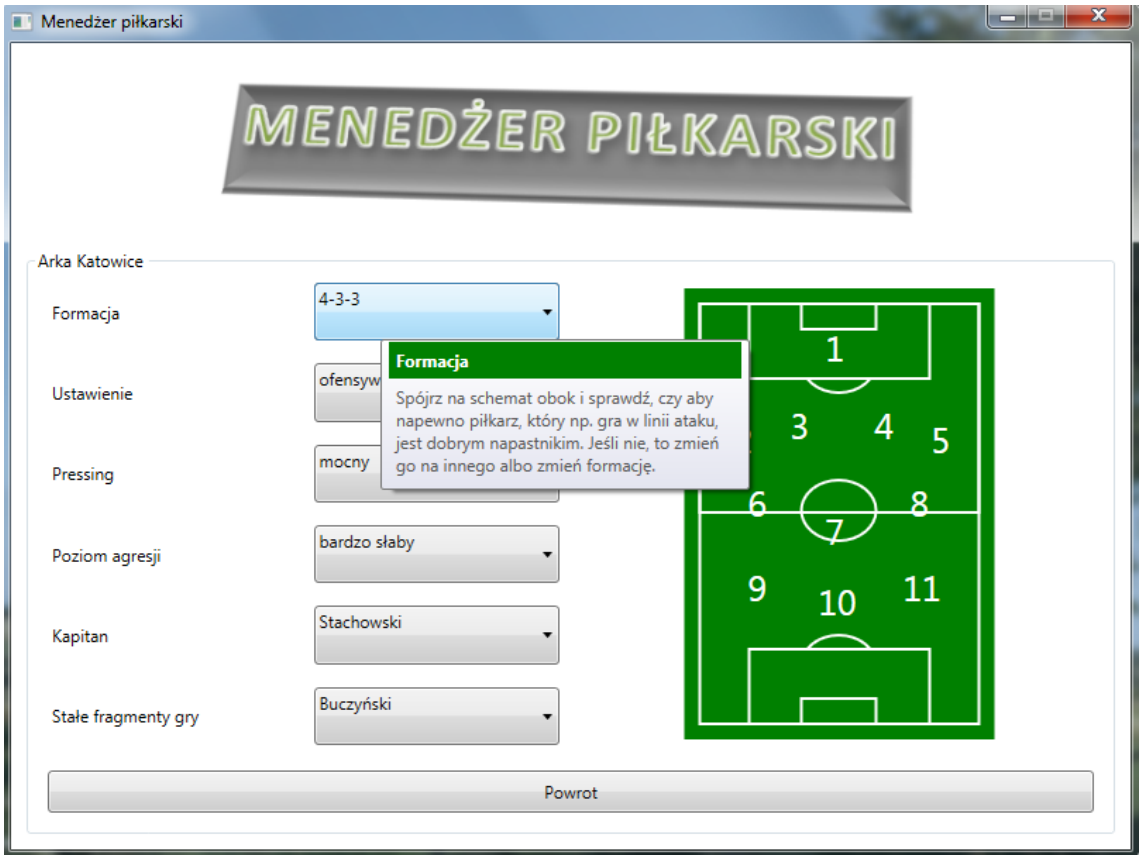
decyzyjne (rysunek 3.4). Listing 3.6 przedstawia kod metody odpowiedzialnej za ustalanie pozycji piłkarzy. Następnie obliczane jest, ilu piłkarzy ma ustaloną daną pozycję. W kolejnym kroku odbywa się sortowanie graczy. Aby gra nie była zbyt trudna – jest to sortowanie „odwrócone". Do wyjściowego składu nie dostają sie najlepsi piłkarze. Następnie bilansowane są pozycje piłkarzy. Jeśli na którejś z pozycji jest za mało piłkarzy, przesuwani są zawodnicy z tych, w których są nadmiarowe ilości. Przykładowo, jeśli w drużynie jest 7 obrońców (piłkarzy mających najlepsze umiejętności w obronie) oraz 3 pomocników przy formacji 4-4-2, wtedy jeden z obrońców staje się pomocnikiem. Na sam koniec, w metodzie *ustalNajlepszySklad()* do tablicy obiektów *Pilkarz* przypisywana jest nowa kolejność piłkarzy w składzie. Pierwszych 11 piłkarzy spośród 20 będzie uczestniczyło w następnym meczu.

3.8 Założenia taktyczne

Oprócz ustalania pozycji dla piłkarzy, gracz może ustalać założenia taktyczne. Najważniejszą kwestią w taktyce jest *formacja*. Dzięki niej możemy optymalnie wykorzystać ustawienie piłkarzy. Właśnie od pozycji, na której gra piłkarz, zależy ile wnosi do drużyny. Przykładowo, jeśli w linii obrony mamy piłkarza z niskim poziomem defensywy, obniża to poziom obrony drużyny. Należy zmienić go na piłkarza mającego lepsze umiejętności defensywne albo zmienić formację, na taką w której jest mniej obrońców. Do wyboru jest 7 formacji: *5-4-1, 5-3-2, 4-5-1, 4-4-2, 4-3-3, 3-5-2, 3-4-3*.

Listing 3.6: Metoda *ustalNajlepszePozycjeDlaPilkarzy* klasy *Menedzer*

```
void ustalNajlepszePozycjeDlaPilkarzy()
{
  for (int i = 0; i < klub.iloscPilkarzy; i++)
  {
    if (klub.pilkarze[i].umBramkarskie > klub.pilkarze[i].skutecznosc)
    {
      if (klub.pilkarze[i].umBramkarskie > klub.pilkarze[i].defensywa)
      {
        if (klub.pilkarze[i].umBramkarskie > klub.pilkarze[i].rozgrywanie)
          klub.pilkarze[i].najlepszaPozycja = Pozycja.bramkarz;
        else
          klub.pilkarze[i].najlepszaPozycja = Pozycja.pomocnik;
      }
      else
      {
        if (klub.pilkarze[i].defensywa > klub.pilkarze[i].rozgrywanie)
          klub.pilkarze[i].najlepszaPozycja = Pozycja.obronca;
        else
          klub.pilkarze[i].najlepszaPozycja = Pozycja.pomocnik;
      }
    }
    else
    {
      if (klub.pilkarze[i].skutecznosc > klub.pilkarze[i].defensywa)
      {
        if (klub.pilkarze[i].skutecznosc > klub.pilkarze[i].rozgrywanie)
          klub.pilkarze[i].najlepszaPozycja = Pozycja.napastnik;
        else
          klub.pilkarze[i].najlepszaPozycja = Pozycja.pomocnik;
      }
      else
      {
        if (klub.pilkarze[i].defensywa > klub.pilkarze[i].rozgrywanie)
          klub.pilkarze[i].najlepszaPozycja = Pozycja.obronca;
        else
          klub.pilkarze[i].najlepszaPozycja = Pozycja.pomocnik;
      }
    }
  }
}
```

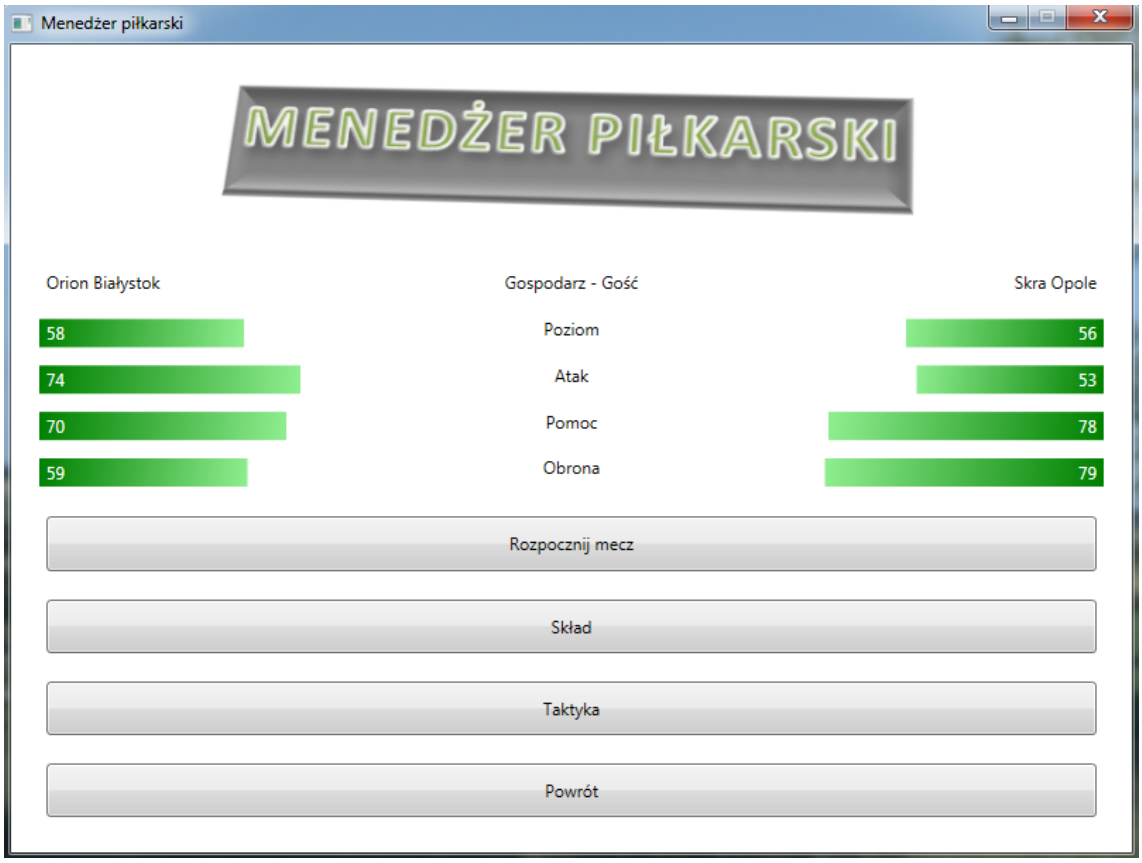


Rysunek 3.5: Screen z aplikacji przedstawiający taktykę drużyny

Gdy ustalimy już podstawową jedenastkę piłkarzy i formację, powinniśmy dopasować *ustawienie*. Możemy wybierać spośród 5 wariantów: *bardzo defensywne, defensywne, neutralne, ofensywne, bardzo ofensywne*. W zależności od poziomów obrony, pomocy i ataku wybieramy odpowiednie ustawienie. Jeśli nasza drużyna jest silna w obronie, ale słaba w ataku, wtedy możemy wybrać ustawienie bardziej ofensywne, aby zbilansować poziomy drużyny. Spowoduje to wzmocnienie ataku i osłabienie obrony.

> *Pressing – sposób gry w piłce nożnej. Polega na naciskaniu przeciwnika posiadającego piłkę, nie pozwalając jemu na rozgrywanie akcji. Pressing jest wymagającym sposobem gry – piłkarze grający nim muszą posiadać silną kondycję.*

Pressing wpływa na posiadanie piłki. Im wyższy ustalimy, tym lepsza powinna być pomoc drużyny. Jednak nasi pomocnicy powinni mieć mocną kondycję. Inaczej zamiast wyższego posiadania piłki możemy mieć niższe. *Poziom agresji* drużyny wpływa na umiejętności linii obrony. Podobnie tak jak w pressingu, tutaj też jest coś w zamian. Im mocniejsza agresja, tym większe prawdopodobieństwo podyktowania



Rysunek 3.6: Screen z aplikacji przedstawiający poziomy drużyn

rzutu karnego dla przeciwnika. W przypadku pressingu i poziomu agresji gracz ma do wyboru 5 możliwości: *bardzo słaby, słaby, zwykły, mocny, bardzo mocny.*

Kapitan jest najważniejszym piłkarzem zespołu. Powinien posiadać wysokie umiejętności przywódcze. Ich poziom wpływa na siłę linii pomocy. Natomiast wykonawca stałych fragmentów gry wykonuje rzuty karne. Warto, żeby był dobrym egzekutorem jedenastek. Na szansę wykorzystania rzutu karnego wpływa umiejętność *Strzały.* Oczywiście im wyższy poziom, tym lepiej. Wszystkie wymienione opcje ustalamy w rozwijanych listach w oknie Taktyka (rysunek 3.5).

3.9 Obliczanie poziomów drużyn

Za obliczanie poziomów drużyn odpowiada klasa *Klub.* Zawiera ona wszelkie informacje o klubie: od statystyki poziomów, statystyk ligowych, taktyki po tablice meczów, piłkarzy, pierwszej jedenastki czy poszczególnych pozycji. Na listingu 3.7 znajduje się fragment metody *obliczPoziomy.*

W pierwszej kolejności zerowane są poszczególny poziomy. Następnie zliczane są ilości obrońców, pomocników i napastników. Każdy klub posiada ustaloną for-

Listing 3.7: Metoda *obliczPoziomy* klasy *Klub*

```
public void obliczPoziomy()
{
    ...
    int iloscObroncow = (int)char.GetNumericValue(formacja[0]);
    int iloscPomocnikow = (int)char.GetNumericValue(formacja[2]);
    int iloscNapastnikow = (int)char.GetNumericValue(formacja[4]);
    // OBLICZANIE POZIOMU OBRONY
    int tym = 0;
    int k = 1;
    int n = 1 + iloscObroncow;
    for (int i = k; i < n; i++)
    {
        if (i < iloscPilkarzy)
            tym += pilkarze[i].defensywa;
    }
    tym += pilkarze[0].umBramkarskie;
    obrona = tym / (iloscObroncow + 1);
    ...
}
```

mację przetrzymywaną w zmiennej string *formacja*. Przykładowo dla taktyki 4-4-2, zapisana jest ona jako *4-4-2*. Za pomocą funkcji *GetNumericValue* „wydobywane" są poszczególne znaki napisu. Ilość obrońców jest zerowym znakiem, pomocników drugim, a napastników czwartym. Następnie obliczane są poziomy obrony, pomocy, ataku, kondycji. Pod koniec dodatkowo sumowane lub odejmowane są punkty za ustawienie, pressing, poziom agresji i poziom przywództwa kapitana drużyny.

3.10 Symulacja meczu

Bardzo ważną częścią gry menedżera piłkarskiego jest symulacja meczów. W aplikacji odpowiadają za to klasy *Mecz, Akcja* oraz *Zdarzenie*. W nich znajdują się wszelkie informacje o danym meczu (rysunek 3.7). Wynik rozegranego meczu zależy od trzech poziomów statystyk każdej z drużyn: *atak*, *pomoc*, *obrona*. Oprócz tego jest również *poziom*, który jest uogólnioną średnią umiejętności wszystkich piłkarzy. W przeciwieństwie do poprzednich trzech statystyk nie wpływa bezpośrednio na wynik rozgrywanego meczu, a jedynie ma orientacyjnie ukazywać graczowi średni poziom wszystkich piłkarzy w drużynie. O sile ataku, pomocy i obrony stanowi pierwsza jedenastka klubu. Na poziom ataku mają wpływ napastnicy, pomocy – pomocnicy,

a obrony – obrońcy i bramkarz.

Przed rozpoczęciem meczu porównywane są poziomy umiejętności obu drużyn. Screen z aplikacji przedstawiający poziomy drużyn znajduje się na rysunku 3.6. Na podstawie porównań obliczane są różnice pomiędzy konkretnymi formacjami (listing 3.8). Pomoc gospodarza jest porównywana z pomocą gościa. Z tego wyliczane jest orientacyjne posiadanie piłki – *posiadaniePilkiGospodarzOgolnie* i *posiadaniePilkiGoscOgolnie*. Siła ataku gospodarza porównywana jest z poziomem obrony gościa. Na podstawie tego obliczana jest *skutecznoscAtakuGospodarzOgolnie*. Analogicznie wyliczana jest *skutecznoscAtakuGoscOgolnie*.

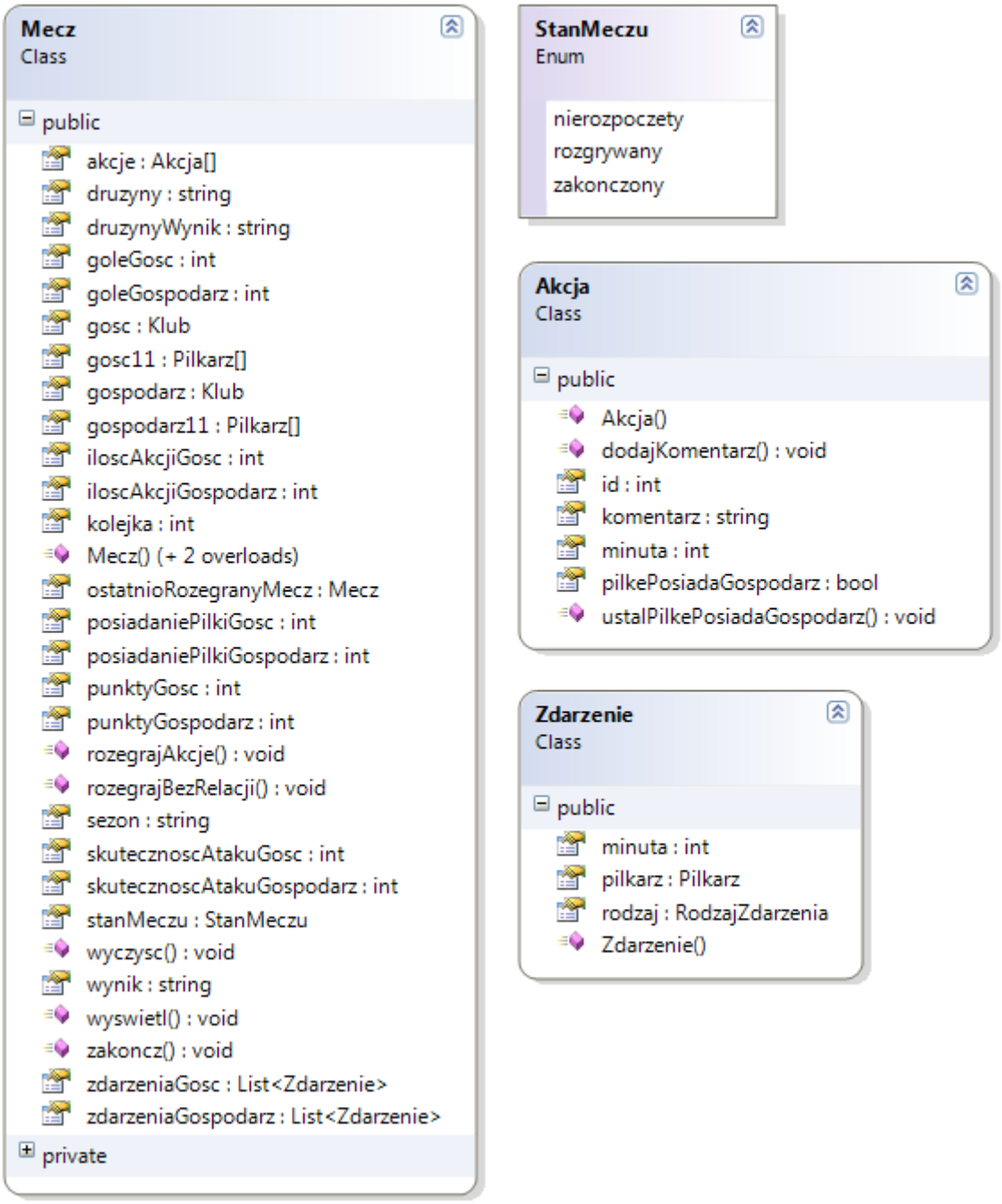
Wszystkie przedstawione powyżej wyliczenia są ogólne. Na podstawie ich odbywa się właściwa symulacja meczu. Każdy mecz trwa „90 minut" (w aplikacji 90 sekund z możliwością przyspieszenia). W tym czasie rozgrywanych jest 90 akcji. W każdej minucie meczu, na podstawie ogólnego posiadania piłki, losowane jest kto posiada w danej chwili piłkę. Przykładowo, jeśli drużyna ma ogólne posiadanie piłki 55%, to ma tyle samo procent szans na to, że w tej minucie posiada piłkę. Gdy dany zespół jest w posiadaniu piłki, to na podstawie skuteczności ataku losowane jest zdobycie gola. Na przykład, jeśli ogólna skuteczność ataku drużyny posiadającej piłkę wynosi 5%, to właśnie tyle jest procent szans na to, że zostanie zdobyty gol przez atakujący zespół w danej akcji.

Oprócz tego, istnieje tzw. zdarzenie specjalne – rzut karny. Szansa na wystąpienie rzutu karnego w danej akcji zależy od poziomu agresji drużyny nie mającej piłki (broniącej się). Rzut karny może wystąpić w tej samej minucie co gol z akcji. Aby obliczyć szanse na strzelenia gola, porównywane są umiejętności strzeleckie wykonawcy rzutu karnego z umiejętnościami bramkarskimi piłkarza ustawionego na pozycji bramkarza. Na podstawie tych porównań wyliczana jest różnica poziomów, a z niej szanse na wykorzystanie rzutu karnego.

3.11 Tryb menedżera – rozgrywki ligowe

W przypadku rozegrania meczu towarzyskiego gracz ma do dyspozycji wybór składu, taktyki oraz samą symulację meczu. Natomiast w trybie menedżera gracz może rozegrać całą serię 30 spotkań w sezonie. Mierzy się z drużynami o różnych poziomach umiejętności. Gracz może sprawdzać terminarze spotkań, analizować tabelę ligową i przygotowywać odpowiednią taktykę na każdą drużynę. Jest to znaczne uatrakcyjnienie gry, pozwalające na konkurowanie z innymi drużynami o zajęcie jak najlepszego miejsca w lidze.

W trybie menedżera możemy rozegrać pełny sezon ligowy składający się z 30



Rysunek 3.7: Diagramy klas *Mecz, Akcja, Zdarzenie* i typu enum *StanMeczu*

Listing 3.8: Metoda *obliczRoznicePoziomow* klasy *Mecz*

```
void obliczRoznicePoziomow()
{
  roznicaPomoc = (gospodarz.pomoc - gosc.pomoc) / 10;
  roznicaAtakGospodarzObronaGosc = (gospodarz.atak - gosc.obrona) / 10;
  roznicaAtakGoscObronaGospodarz = (gosc.atak - gospodarz.obrona) / 10;
  // OBLICZANIE OGÓLNEGO POSIADANIA PIŁKI PRZEZ GOSPODARZA W PROCENTACH
  if (roznicaPomoc ==  0) { posiadaniePilkiGospodarzOgolnie = 50; }
  if (roznicaPomoc ==  1) { posiadaniePilkiGospodarzOgolnie = 53; }
  ...
  posiadaniePilkiGoscOgolnie = 100 - posiadaniePilkiGospodarzOgolnie;
  // OBLICZANIE SKUTECZNOSCI ATAKU GOSPODARZA W PROMILACH
  if (roznicaAtakGospodarzObronaGosc == 0) skutecznoscAtakuGospodarzOgolnie = 23;
  if (roznicaAtakGospodarzObronaGosc == 1) skutecznoscAtakuGospodarzOgolnie = 27;
  ...
  // OBLICZANIE SKUTECZNOSCI ATAKU GOSCI W PROMILACH
  if (roznicaAtakGoscObronaGospodarz == 0) skutecznoscAtakuGoscOgolnie = 23;
  if (roznicaAtakGoscObronaGospodarz == 1) skutecznoscAtakuGoscOgolnie = 27;
  ...
}
```

kolejek. Za obsługę rozgrywek ligowych odpowiadają dwie klasy *Liga* i *Kolejka*. Pierwsza z nich, tak jak nazwa wskazuje, jest przedstawieniem rozgrywek ligowych. Istnieje tylko jeden egzemplarz klasy *Liga* – obiekt statyczny dostępny globalnie. Klasa ta zawiera w sobie m. in. wszystkie kluby oraz obiekty klasy *Kolejka*. Obydwie instancje obiektów są przechowywane w tablicach. Tablica drużyn zawiera 16 obiektów klasy *Klub*. Natomiast tablica kolejek zawiera 30 egzemplarzy obiektów klasy *Kolejka*. *Liga* zawiera m. in. następujące metody:

- *rozegrajKolejke*
- *uaktualnijStatystyki*
- *wyczyscStatystykiLigowe*
- *sprawdzMiejsce*
- *sprawdzOstatnie5Meczow*
- *sprawdzPoprzedniMecz*
- *sprawdzNastepnyMecz*

Listing 3.9: Metoda *rozegrajKolejke* klasy *Liga*

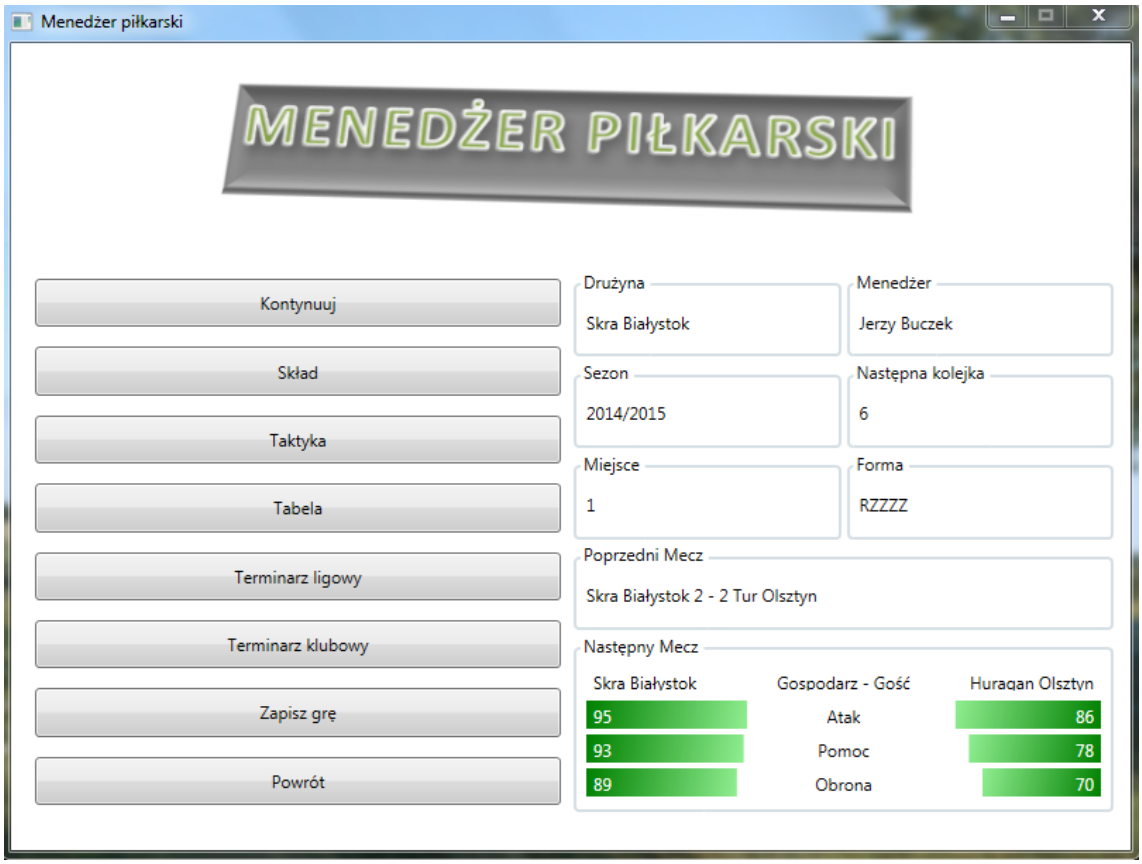
```
public void rozegrajKolejke()
{
  if (nrKolejki < iloscKolejek)
  {
    kolejki[nrKolejki].rozegraj();
    nrKolejki++;
    sprawdzMiejsce();
  }
}
```

Listing 3.10: Metoda *rozegraj* klasy *Kolejka*

```
public void rozegraj()
{ for (int i = 0; i < iloscMeczow; i++) mecze[i].rozegrajBezRelacji(); }
```

Metody zaczynające się na *sprawdz* wykorzystywane są przede wszystkim w oknie rozgrywek ligowych, gdzie możemy sprawdzać poprzedni mecz, następny mecz, formę drużyny czy miejsce klubu w tabeli ligowej.

Natomiast każdy obiekt klasy *Kolejka* zawiera w sobie tablicę 8 meczów. W konstruktorze znajdują się m. in. definicje poszczególnych meczów. Oprócz tego, *Kolejka* zawiera dwie metody: *rozegraj* i *uaktualnijStatystyki*. Jak widać mają takie same bądź zbliżone nazwy do funkcji klasy *Liga*. Jest tak, ponieważ *Liga* pełni funkcję „pojemnika" na kolejki i jest klasą pośredniczącą między klasą wywołującą metodę a klasą *Kolejka*. Na listingu 3.10 jest metoda, która jest wywoływana przez funkcję znajdującą się na listingu 3.9.



Rysunek 3.8: Screen z aplikacji przedstawiający menu

Podsumowanie

W dzisiejszych czasach tworzenie gier jest procesem pracochłonnym, zajmującym często wiele lat. Wielkie produkcje są tworzone przez zespoły różnych specjalistów. Tworzenie gier przez jedną osobę wiąże się z zajmowaniem się wszystkimi zagadnieniami, które w zespołach są przeprowadzane przez różne osoby. Dziś rzadko się zdarza, że gry są pisane przez pojedynczych autorów. Jeśli tak jest, zazwyczaj są to tytuły dość proste. Oryginalność jest czynnikiem, który pozwala im stać się popularnymi grami.

Stworzenie tej gry wymagało ode mnie zajęciem się częścią logiczną aplikacji jak i również graficznym interfejsem użytkownika. Praktycznie całą logikę gry napisałem sam. Natomiast tworząc graficzny interfejs użytkownika, korzystałem z gotowych kontrolek systemu *Windows*. Napisanie tej aplikacji zajęło mi dużo czasu. Jest to największy projekt spośród, tych które napisałem dotychczas. Stworzenie tej gry było dość ciekawym zagadnieniem. Pozwoliło mi rozwinąć moje umiejętności oraz nabyć trochę doświadczenia w dziedzinie inżynierii oprogramowania. Uważam że tworzenie aplikacji, a zwłaszcza gier jest bardzo interesującym i wciągającym zajęciem.

W przyszłości, pracując już hobbistycznie chciałbym rozwinąć grę – dodać kolejne funkcje menedżera piłkarskiego. Myślałem o tym, aby opublikować tę grę (bardziej rozwiniętą wersję) na systemy mobilne, tj. Android. Zapewne wiązałoby się to z przepisaniem aplikacji na język Java. Oprócz tego, zastanawiam się nad stworzeniem prostej gry zręcznościowej, również związanej z tematyką piłki nożnej. W przyszłości mógłbym połączyć obie aplikacje, tworząc pełną grą piłkarską – połączenie gry strategicznej ze zręcznościową: zarówno pozwalającą na bycie menedżerem oraz sterowanie piłkarzami w czasie meczu.

Literatura

[1] *Słownik języka polskiego PWN*, Wydawnictwo naukowe PWN, Warszawa 1995

[2] Jarosław Cisek, *Tworzenie nowoczesnych aplikacji graficznych w WPF*, Helion, 2012

[3] Ian Griffiths, Matthew Adams, Jesse Liberty, *C#. Programowanie. Wydanie VI*, Helion, 2012

[4] Marcin Lis, *SQL Ćwiczenia praktyczne*, Helion, Gliwice 2011

[5] Zbigniew Palka, Andrzej Ruciński, *Wykłady z kombinatoryki*, Wydawnictwa Naukowo-Techniczne, Warszawa 2004

[6] *https://cloud.blender.org/training/3d-printing/* [dostęp 30.06.2015]

[7] *https://durian.blender.org/about/* [dostęp 29.06.2015]

[8] *https://msdn.microsoft.com/pl-pl/library/ms123401.aspx* [dostęp 26.07.2015]

[9] *http://max3d.pl/artykuly/40/wywiad-tomasz-baginski-pierwszy-wywiad-dla-max3d* [dostęp 29.06.2015]

[10] *http://www.ppe.pl/news-24642-nad_gta_v_pracowalo_ponad_1000_osob.html* [dostęp 27.06.2015]

[11] *http://www.redbull.com/pl/pl/games/stories/1331708755539/historia-football-manager* [dostęp 02.07.2015]

[12] *http://www.wsjp.pl/* [dostęp 02.07.2015]

Spis rysunków

Spis listingów

Załącznik – płyta CD

Na płycie CD znajdują się 4 katalogi:

- Dokumentacja
- Kody źródłowe aplikacji
- Literatura – strony WWW
- Postać wykonywalna aplikacji

W folderze *Dokumentacja* znajduje plik z rozszerzeniem *pdf* oraz katalog *Pliki źródłowe*. Plik pdf jest pracą dyplomową w wersji elektronicznej. Natomiast w katalogu znajduje się plik z rozszerzeniem *tex*, który jest dokumentem źródłowym wersji pdf. Jest to plik, który można modyfikować. Wraz z nim znajdują się m. in. obrazy wykorzystywane w dokumencie oraz pliki utworzone przez system składu tekstu LaTeX.

Aplikacja może nie łączyć się z bazą danych, jeśli nie jest zainstalowany *Microsoft Office (Access)* przynajmniej wersja 2007 oraz jeśli Office jest w wersji 64 bitowej. Jest możliwość zainstalowania samego silnika bazy danych bez instalacji pakietu Office. Można pobrać go ze strony Microsoftu.

https://www.microsoft.com/en-us/download/details.aspx?id=13255

[dostęp 22.07.2015]

W kodach źródłowych aplikacji znajduje się zapisany projekt stworzony w środowisku *Microsoft Visual C# 2010 Express*. Strony internetowe, które podane są w literaturze, zostały zapisane w katalogu *Literatura – strony WWW*. Natomiast plik wykonywalny programu znajduje się w folderze *Postać wykonywalna aplikacji*.